VARIABLES AMBIENTALES DE CIÉNAGAS.

Software para la transmisión y recepción de datos de sensores de pH, potencial redox, oxígeno disuelto, conductividad eléctrica y temperatura.

MANUAL TÉCNICO.

Abril 2024.

VAMCI

VARIABLES AMBIENTALES DE CIÉNAGAS.

UNIVERSIDAD DE ANTIOQUIA
1 8 0 3

Facultad de INGENIERÍA

GeoLimna

GEPAR U. de A.

Tabla de contenido.

1. INTRODUCCIÓN.

Este proyecto busca automatizar la medición de variables acuáticas como el pH, potencial redox, oxígeno disuelto, conductividad eléctrica y temperatura, subiendo automáticamente los datos a una base en la nube a través de Google Firebase. Esto permite que los datos sean accesibles desde cualquier dispositivo inteligente mediante una aplicación móvil creada en Kodular. Este sistema ayuda a reducir los errores de medición manual. Aunque está diseñado para la ciénaga de Ayapel en Córdoba, el sistema puede adaptarse y escalar para usarse en otros humedales o cuerpos de agua similares.

El diseño incluye dos microcontroladores y dos dispositivos que transmiten datos usando radiofrecuencia y conexiones seriales e inalámbricas. Consiste en un punto remoto y otro fijo, este último en un centro de datos donde los datos se monitorean, almacenan y utilizan para crear modelos predictivos de inundaciones y otros eventos climáticos.

Se emplea software Arduino y programación por bloques para el desarrollo de la aplicación. Los dispositivos deben ser calibrados para asegurar mediciones precisas. Se explicará cómo conectar todas las etapas del proyecto correctamente para que el sistema funcione de manera adecuada.

2. OBJETIVOS.

2.1. Objetivo general

Desarrollar y ensamblar un sistema que facilite la transmisión y el almacenamiento de información de sensores a distancia para el proyecto VAMCI bajo el código SIIU 2020-37870, con el objetivo de visualizarlos en la aplicación web GEOVAM y la aplicación móvil correspondiente, y utilizarlos más adelante en el desarrollo de modelos predictivos.

2.2. Objetivos específicos

- Identificar un nuevo grupo de variables ambientales que serán registradas por dispositivos electrónicos de uso comercial.
- Incorporar sensores adicionales a la estructura preexistente de VAMCI diseñando la transmisión de datos mediante dispositivos LoraWAN hacia un archivo de tipo .csv.
- Combinar los datos obtenidos con los ya presentes en la base de datos en tiempo real del servidor de Google usando el protocolo de Firebase, para luego almacenarlos en Google Sheets y visualizarlos en GEOVAM, accesibles en cualquier momento y desde cualquier lugar del mundo.
- Crear una aplicación móvil simple en Kodular, aún no existente, que permita acceder a los datos del sistema en tiempo real desde cualquier dispositivo inteligente.

3. REQUIRIMIENTOS TÉCNICOS.

La implementación y operación efectiva del Sistema remoto para la medición y el registro de variables ambientales se realizó empleando diversos dispositivos y utilizando software con las características detalladas a continuación.

3.1. Hardware.

- Sensores Atlas Scientific: pH, Potencial Redox (ORP), Oxígeno Disuelto (OD), Conductividad Eléctrica (EC), Temperatura (RTD).
- Placa Whitebox T2.
- Arduino Mega 2560.
- 2 módulos Lora WiFi V2 (transmisor y receptor).
- NodeMCU 1.0
- 2 fuentes suicheadas 5V 3A 15W
- Cables encauchetados resistente al agua.
- Cables Macho-Macho para Arduino.
- Cable UTP.
- 2 cajas plásticas con tapa.

- Boards: una microboard y una convencional.
- Cinta doble faz.
- 2 cajas organizadoras de verduras.
- Tubos de PVC para dirigir las puntas de prueba de los sensores.
- Correas fijadoras plásticas.

3.2. Software.

- Arduino Software (IDE).
- Firebase.
- App Script (Google Sheets).
- GEOVAM (Aplicativo de visualización).
- Kodular (Diseño apps móviles).

4. CONTEXTUALIZACIÓN TÉCNICA DEL PROBLEMA.

Se describen a continuación las características del problema al que se le dio solución.

- El principal desafío fue establecer la conexión de los sensores subacuáticos con un punto de acceso para consultar y controlar la información, vital para diseñar modelos de predicción, evitando poner en riesgo a las comunidades de lugares geográficamente apartados como lo es Ayapel.
- Se logró comunicar el punto remoto en el agua con el centro de datos, con un alcance de hasta dos kilómetros, mediante los módulos Lora WiFi y una programación exitosa.
- Se requería un dispositivo como el NodeMCU para poder compartir esta información en la base de datos en tiempo real y dirigirla hacia el repositorio en Google Sheets, permitiendo la visualización de cada histórico de las variables mencionadas en GEOVAM.
- También era necesario asegurar que el punto remoto tuviera la flotabilidad adecuada para evitar que se hundiera. Estas pruebas se llevaron a cabo en la piscina de la Universidad de Antioquia, donde se confirmó este aspecto.
- Una consideración adicional importante era la alimentación de energía de los puntos. Para ello, se obtuvo un rendimiento óptimo utilizando fuentes de voltaje conmutadas de 5V y 3A, lo que garantiza un funcionamiento exitoso siempre que haya acceso a la red eléctrica.

5. DOCUMENTACIÓN DEL CÓDIGO.

El desarrollo del sistema requirió la creación de cuatro códigos en el entorno de desarrollo integrado (IDE) de Arduino, los cuales gestionan las comunicaciones entre los distintos dispositivos. Además, se empleó el entorno de desarrollo de Kodular para diseñar y desarrollar la aplicación denominada VAMCI.

5.1. Código IDE Arduino.

Los cuatro dispositivos mencionados fueron programados en el entorno de desarrollo integrado (IDE) de Arduino para adquirir, comunicar y transmitir los datos de los sensores. Estos datos son llevados al transmisor Lora, transmitidos al punto estático y luego transferidos al NodeMCU, que se conecta a una red local WiFi e inserta los datos en la base de datos en tiempo real. Posteriormente, estos datos son almacenados y graficados en GEOVAM, su destino final. Cada instrucción será detallada con una explicación correspondiente para una mejor comprensión.

Se debe mencionar que los sensores ya han sido calibrados previamente y están configurados en modo I2C por conveniencia, aunque también es posible calibrarlos en modo serial si así se desea. El proceso de calibración se encuentra disponible en el siguiente enlace, donde se detallan los pasos necesarios para llevar a cabo la calibración.

https://atlas-scientific.com/embedded-solutions/ezo-ph-circuit/

En la sección izquierda, dar clic en *Documents & Downloads*, luego seleccionar *UART and I2C mode switching* donde se dirige a este sitio:

https://www.instructables.com/UART-AND-I2C-MODE-SWITCHING-FOR-ATLAS-SCIENTIFIC-E/

Nota: De la misma manera, se calibran los demás sensores.

5.1.1. Adquisición de datos desde los sensores mediante placa Whitebox ajustable al Arduino Mega 2560.

Para adquirir datos de los sensores Atlas Scientific, es necesario incluir las bibliotecas Ezo_12c.h y Wire.h. La primera facilita el acceso y el control de los datos medidos por los sensores, mientras que la segunda permite la comunicación con los sensores a través del protocolo I2C mediante la placa Whitebox y el Arduino Mega 2560. Una vez en Arduino, los datos se enviarán al Lora Transmisor a través del puerto serial. Se asignan los pines 19 como receptor y 18 como transmisor en el Arduino Mega. Se crean

objetos con direcciones correspondientes para cada sensor, y se define la variable Datos para almacenar los datos capturados por el puerto I2C.

```
#include <Ezo_i2c.h>
#include <Wire.h>
#include <SoftwareSerial.h>

int StateSensors = HIGH;
String Datos = "";
SoftwareSerial mySerial(19, 18);

Ezo_board orp = Ezo_board(98, "ORP");
Ezo_board od = Ezo_board(97, "DO");
Ezo_board ph = Ezo_board(99, "PH");
Ezo_board ec = Ezo_board(100, "EC");
Ezo_board rtd = Ezo_board(102, "TEMP");
```

Se configura la velocidad a la que se enviarán los datos por el puerto serial y se inicia la comunicación serial.

```
void setup() {
  Wire.begin();
  mySerial.begin(9600);
  Serial.begin(9600);
}
```

Se inicializa un bucle que se ejecutará continuamente, donde la primera acción es leer el valor de cada sensor registrado a través del puerto I2C.

```
void loop() {

  ph.send_read_cmd();
  rtd.send_read_cmd();
  ec.send_read_cmd();
  orp.send_read_cmd();
  od.send_read_cmd();
  delay(2000);
```

Si los sensores tienen datos medidos, se obtiene la lectura de esos valores y se concatenan en la variable "Datos". Luego, se muestra el mensaje "Mensaje a enviar al Lora TX:" junto con el valor correspondiente. Este mensaje se envía a través del puerto serial y la variable "Datos" se limpia, dejándola disponible para la próxima lectura de otro sensor en futuras iteraciones. Este proceso se repite hasta completar las lecturas y envíos de los cinco sensores: Temperatura (RTD), Potencial Redox (ORP), Conductividad eléctrica (EC), Oxígeno Disuelto (OD) y pH.

```
  // Si el pulsador está presionado
    if (StateSensors == HIGH)
    {
      Datos = receive_reading(rtd);
      Datos= "RTD/"+Datos+"\n";
      Serial.print("Mensaje a enviar al LORA TX: ");
      Serial.println(Datos);
      writeString(Datos);
      Datos="";

      Datos = receive_reading(orp);
      Datos= "ORP/"+Datos+"\n";
      Serial.print("Mensaje a enviar al LORA TX: ");
      Serial.println(Datos);
      writeString(Datos);
      Datos="";

      Datos = receive_reading(ec);
      Datos= "EC/"+Datos+"\n";
      Serial.print("Mensaje a enviar al LORA TX: ");
      Serial.println(Datos);
      writeString(Datos);
      Datos="";

      Datos = receive_reading(od);
      Datos= "OD/"+Datos+"\n";
      Serial.print("Mensaje a enviar al LORA TX: ");
      Serial.println(Datos);
      writeString(Datos);
      Datos="";

      Datos = receive_reading(ph);
      Datos= "PH/"+Datos+"\n";
      Serial.print("Mensaje a enviar al LORA TX: ");
      Serial.println(Datos);
      writeString(Datos);
      Datos="";
    }
}
```

Dado que todos los datos se han concatenado, se emplea la función writeString para enviar la cadena de caracteres a través del puerto serial. Sin embargo, en cada paso de este bucle, se envía cada carácter individualmente. Además, se espera un intervalo de tres segundos entre cada iteración para evitar superposiciones en la información que se envía.

```
void writeString(String stringData) {
  for (int i = 0; i < stringData.length(); i++)
  {
    mySerial.write(stringData[i]);

    if(stringData[i]=='\n'){
      delay(2000);
    }
  }

  delay(1000);
}
```

Finalmente, se implementa una función para decodificar la lectura después de enviar el comando de lectura de los datos. Se imprime primero el comando de lectura, luego se obtienen los datos y se utiliza una estructura de selección switch-case. Si el comando es exitoso, se imprime la lectura; si el comando falla, se muestra un mensaje de alerta: "Failed"; si el comando aún está en proceso de medición, se imprime el mensaje: "Pending"; y si el sensor no tiene datos para enviar, se imprime: "No Data".

```
String receive_reading(Ezo_board &Sensor) {
  String Data="";

  Serial.print(Sensor.get_name()); Serial.print(": ");
  Sensor.receive_read_cmd();
  switch (Sensor.get_error()) {
    case Ezo_board::SUCCESS:
      Serial.println(Sensor.get_last_received_reading());
      Data=Sensor.get_last_received_reading();
      return Data;
      break;

    case Ezo_board::FAIL:
      Serial.print("Failed ");
      break;

    case Ezo_board::NOT_READY:
      Serial.print("Pending ");
      break;

    case Ezo_board::NO_DATA:
      Serial.print("No Data ");
      break;
  }
}
```

5.1.2. Transmisión serial de datos entre el Arduino Mega 2560 y el Heltec Lora WiFi transmisor.

Para recibir los datos del Arduino Mega 2560, se deben incluir las mismas bibliotecas mencionadas anteriormente, además de integrar las bibliotecas LoRaWan_APP.h para habilitar las funciones de los módulos Lora Heltec WiFi, stdlib.h y HardwareSerial.h, que proporcionan los prototipos de funciones y las capacidades de comunicación serial (UART) para Arduino.

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#include <Ezo_i2c.h>
#include <Wire.h>
#include "stdlib.h"
#include <HardwareSerial.h>
```

Se crea una etiqueta DEBUG con un bucle que se ejecuta constantemente para detectar errores de forma temprana durante la compilación del código. Se asignan los pines para la comunicación serial, utilizando el pin 6 como receptor y el pin 7 como transmisor, lo que completa la conexión entre el Arduino y el Lora transmisor. También se definen dos variables: "inChar" para leer los caracteres recibidos del Mega, y "string" para almacenarlos en memoria dinámica.

```
#define DEBUG(a, b)
  for (int index = 0; index < b; index++)
    Serial.print(a[index]);
    Serial.println();
#define rx 6
#define tx 7
HardwareSerial serial1(1);

//Definición de Variables
char inChar;
String string="";
```

Se realiza la configuración de parámetros para la transmisión LoRa como frecuencia (RF_FREQUENCY), potencia de salida (TX_OUTPUT_POWER), entre otros.

Además, Se definen los Buffers (txpacket y rxpacket) para almacenar los paquetes de datos a enviar y recibir y la variable lora_idle para gestionar el estado del dispositivo LoRa, indicando si está listo para enviar otro paquete.

```
#define RF_FREQUENCY                                915000000 // Hz
#define TX_OUTPUT_POWER                             5        // dBm
#define LORA_BANDWIDTH                              0         // [0: 125 kHz,
#define LORA_SPREADING_FACTOR                       7         // [SF7..SF12]
#define LORA_CODINGRATE                             1         // [1: 4/5,
#define LORA_PREAMBLE_LENGTH                        8         // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT                         0         // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON                  false
#define LORA_IQ_INVERSION_ON                        false
#define RX_TIMEOUT_VALUE                            1000
#define BUFFER_SIZE                                 80//Define the payload size here

char txpacket[BUFFER_SIZE];
char rxpacket[BUFFER_SIZE];
double txNumber;
bool lora_idle=true;
static RadioEvents_t RadioEvents;

void OnTxDone( void );
void OnTxTimeout( void );
```

Luego, se inicializa la comunicación serial, se configuran los parámetros del radio LoRa (como la frecuencia, potencia de salida, ancho de banda, etc.) utilizando la función Radio.SetTxConfig(), y prepara los eventos de transmisión (OnTxDone y OnTxTimeout) para manejar el resultado de las operaciones de envío.

```
void setup() {
    serial1.begin(9600, SWSERIAL_8N1,rx,tx);
    Serial.begin(9600);
    Mcu.begin();
    string.reserve(200);
    txNumber=0;

    RadioEvents.TxDone = OnTxDone;
    RadioEvents.TxTimeout = OnTxTimeout;
    Radio.Init( &RadioEvents );
    Radio.SetChannel( RF_FREQUENCY );
    Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
                                   LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                                   LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                                   true, 0, 0, LORA_IQ_INVERSION_ON, 3000 );
}
```

Teniendo listo lo anterior, se inicializa la comunicación serial y se verifica si el puerto serial está disponible. Entonces, se inicia la

lectura de caracteres con una verificación del final de trama: si el carácter leído es diferente de "\n", se concatenan todos los caracteres anteriores en la variable "string". Si el carácter leído es igual a "\n", se imprimen en la consola todos los caracteres leídos hasta el final de la trama.

```
void loop()
{
  if(lora_idle == true)
  {
    if (serial1.available()){
      inChar = serial1.read();
      if(inChar!='\n'){
         string+=inChar;
         Serial.println(string);
      }
      Serial.println(string);
      if(inChar=='\n'){
        string+=inChar;
        Serial.println("Lectura sensores: ");
        Serial.println(string);
```

5.1.3. Transmisión de datos por radiofrecuencia entre el Heltec Lora transmisor y el Heltec Lora receptor.

Después de leer los datos recibidos desde el Arduino Mega a través de la comunicación serial y de concatenar los caracteres en un objeto String hasta encontrar un salto de línea (\n), se completa el mensaje. A continuación, se envía este mensaje a través de la red LoRa utilizando Radio.Send(). Se espera a que el evento de transmisión se complete antes de permitir el envío de otro paquete.

Es crucial definir la banda de transmisión para que los dispositivos LoRa operen en ella, así como definir un paquete para enviar toda la cadena de caracteres con la información de los sensores al receptor LoRa. Finalmente, se configura la comunicación por radiofrecuencia y se envía el paquete completo de datos. El envío se da por concluido una vez que toda la información ha sido transmitida, y se limpia la variable string para su próximo uso en el proceso.

```
void loop()
{
  if(lora_idle == true)
  {
    if (serial1.available()){
      inChar = serial1.read();
      if(inChar!='\n'){
         string+=inChar;
         Serial.println(string);
      }
      Serial.println(string);
      if(inChar=='\n'){
        string+=inChar;
        Serial.println("Lectura sensores: ");
        Serial.println(string);
        sprintf(txpacket,"%s",string);
        Serial.printf("\r\nsending packet \"%s\" , length %d\r\n",txpacket, strlen(txpacket));
        Radio.Send( (uint8_t *)txpacket, strlen(txpacket) ); //send the package out
        string="";
        lora_idle = false;

      }
    }
  }
  Radio.IrqProcess( );
}
```

Se han creado dos eventos **OnTxDone()** y **OnTxTimeout()**, el primero se invoca cuando un paquete se ha transmitido con éxito,

permitiendo que el dispositivo envíe el siguiente paquete y el segundo se invoca si ocurre un timeout durante la transmisión, poniendo el dispositivo LoRa en modo de bajo consumo y permitiendo que el ciclo de envío pueda intentarse de nuevo.

```
void OnTxDone( void )
{
  Serial.println("TX done......");
  lora_idle = true;
}

void OnTxTimeout( void )
{
    Radio.Sleep( );
    Serial.println("TX Timeout......");
    lora_idle = true;
}
```

En el LoRa receptor, se debe ajustar la misma banda de transmisión que se configuró en el LoRa transmisor. También se establece una variable para supervisar la intensidad de la señal de radio más potente disponible, permitiendo así la conexión para la comunicación por radiofrecuencia. Además, se define una variable llamada "packet" que contendrá la trama de datos que se espera recibir.

```
#define RF_FREQUENCY                                915000000 // Hz
#define TX_OUTPUT_POWER                             5        // dBm
#define LORA_BANDWIDTH                              0         // [0: 125 kHz,
#define LORA_SPREADING_FACTOR                       7         // [SF7..SF12]
#define LORA_CODINGRATE                             1         // [1: 4/5,
#define LORA_PREAMBLE_LENGTH                        8         // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT                         0         // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON                  false
#define LORA_IQ_INVERSION_ON                        false
#define RX_TIMEOUT_VALUE                            1000
#define BUFFER_SIZE                                 80 // Define the payload size here

char txpacket[BUFFER_SIZE];   //Se define el paquete el cual va a almacenar los datos recibidos
char rxpacket[BUFFER_SIZE];
```

Se ejecuta un bucle continuamente. Se verifica si el módulo LoRa está en modo idle, y si es así, pone el módulo en modo de recepción (Rx).

```
void loop()
{
  if(lora_idle)
  {
    lora_idle = false;
    Serial.println("into RX mode");
    Radio.Rx(0);
  }
  delay(1000);
  Radio.IrqProcess( );
}
```

Posteriormente, se crea el callback de evento de radio *OnRxDone()*, es una función de callback que se llama cuando se ha recibido un paquete. Guarda el RSSI y el tamaño del paquete, copia el paquete a un buffer y lo imprime por serial.

```
void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr )
{
    rssi=rssi;
    rxSize=size;
    memcpy(rxpacket, payload, size );
    rxpacket[size]='\0';
    Radio.Sleep( );
    Serial.print(rxpacket);
    writeString(rxpacket);
    delay(1000);
    Serial.printf("\r\nreceived packet \"%s\" with rssi %d , length %d\r\n",rxpacket,rssi,rxSize);
    lora_idle = true;
}
```

Finalmente, se desarrolla una función auxiliar llamada writeString() para la escritura en serie. Esta función envía una cadena de caracteres uno por uno a través de serial1 y también la imprime en la consola serial para propósitos de depuración.

```
void writeString(String stringData) {
    for (int i = 0; i < stringData.length(); i++)
    {
      serial1.write(stringData[i]);
      Serial.print(stringData[i]);
    }
}
```

En este punto, los datos se encuentran en el LoRa receptor y están listos para ser enviados al NodeMCU 1.0.

5.1.4. Transmisión serial de datos entre el Heltec Lora receptor y el NodeMCU.

Para establecer la comunicación serial entre el receptor LoRa y el NodeMCU, es necesario agregar las siguientes bibliotecas: PubSubclient.h, que facilita el envío y la recepción de mensajes utilizando el protocolo MQTT; Arduino.h, que proporciona acceso a las constantes, tipos de datos y funciones propias de Arduino; Wifi.h, que brinda soporte para la conexión Wi-Fi, es una biblioteca propia de Arduino; y Firebase_ESP_Client.h, que permite utilizar interfaces de red existentes de clientes de Arduino.
Además, se definen los pines en el receptor LoRa para la transmisión serial, asignando el pin 4 como receptor y el pin 5 como transmisor.

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#include <PubSubClient.h>
#include <WiFi.h>
#include <Firebase_ESP_Client.h>
#include <HardwareSerial.h>

#define DEBUG(a, b)
  for (int index = 0; index < b; index++)
    Serial.print(a[index]);
    Serial.println();
#define rx 4
#define tx 5
HardwareSerial serial1(1);
```

Como ya se había dicho, la función *OnRxDone()* se usa para recibir los datos, e imprimir la cadena de caracteres que fue recibida, pues esa misma cadena de caracteres es la que se debe enviar al NodeMCU.

```
void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr )
{
    rssi=rssi;
    rxSize=size;
    memcpy(rxpacket, payload, size );
    rxpacket[size]='\0';
    Radio.Sleep( );
    Serial.print(rxpacket);
    writeString(rxpacket);
    delay(1000);
    Serial.printf("\r\nreceived packet \"%s\" with rssi %d , length %d\r\n",rxpacket,rssi,rxSize);
    lora_idle = true;
}
```

Se configura la comunicación serial y el módulo LoRa para la recepción de datos, Al llamar a *Radio.SetRxConfig()* se configura detalladamente cómo el LoRa debe manejar la recepción de datos, incluyendo el modo, ancho de banda, factor de esparcimiento, etc. Luego, prepara el evento de recepcion *OnRxDone* para manejar el resultado de las operaciones de datos recibidos.
Finalmente, se llama la función *OnRxDone()*, para que imprima los datos que tiene en cola. Cuando acabe este proceso se usa la función *writeString()* para que escriba por el puerto serie la cadena de caracteres que está en *stringData*. Se envía caracter por caracter hasta la longitud de la cadena.

```
void setup() {
    Serial.begin(9600);
    serial1.begin(9600, SERIAL_8N1,rx,tx);
    Mcu.begin();
    txNumber=0;
    rssi=0;

    RadioEvents.RxDone = OnRxDone;
    Radio.Init( &RadioEvents );
    Radio.SetChannel( RF_FREQUENCY );
    Radio.SetRxConfig( MODEM_LORA, LORA_BANDWIDTH, LORA_SPREADING_FACTOR,
                               LORA_CODINGRATE, 0, LORA_PREAMBLE_LENGTH,
                               LORA_SYMBOL_TIMEOUT, LORA_FIX_LENGTH_PAYLOAD_ON,
                               0, true, 0, 0, LORA_IQ_INVERSION_ON, true );
}
```

.

En el NodeMCU, se asignan los pines de la siguiente manera: el pin RX se establece como 13 y el pin TX como 15. Además, se utiliza una etiqueta DEBUG para detectar errores de forma temprana. Se define la configuración del puerto serie y se declara la variable "inChar" para leer los caracteres que llegan por el puerto serial desde el receptor LoRa (Lora RX).

```
#define DEBUG(a, b)
  for (int index = 0; index < b; index++)
    Serial.print(a[index]);
    Serial.println();
#define rx 13
#define tx 15

char inChar;
```

Si el puerto serial está disponible, se procede a leer los caracteres, concatenándolos en la variable "string" mientras el carácter leído

sea distinto de "\n". Pero si este carácter es "\n", se imprime el mensaje "Dato recibido del LORA punto B:", acompañado de toda la información del sensor recién leída.

```
void loop() {

  if (mySerial.available()){
    inChar = mySerial.read();
    if(inChar!='\n' && inChar!='/'){
       string+=inChar;
    }
    if (inChar=='\n'){
      Serial.print("Dato recibido del LORA punto B: ");
      Serial.print("->");
      Serial.println(string);
```

En este punto, los datos se encuentran en el NodeMCU. A continuación, se explica cómo se lleva a cabo la publicación de estos datos en la base de datos en tiempo real (RTDB) en Firebase.

5.1.5. Transmisión WiFi de datos entre el NodeMCU y publicación de estos en la base de datos en tiempo real en Firebase

Para la publicación de datos en la base de datos en tiempo real (RTDB) de Firebase, se requieren las siguientes bibliotecas: NTPClient.h, que facilita la conexión a un servidor de tiempo NTP para obtener la hora y su sincronización; WiFiUdp.h, que posibilita el envío y recepción de mensajes UDP; addons/TokenHelper.h, que permite proporcionar la información necesaria para acceder al token de la base de datos; y, finalmente, la biblioteca addons/RTDBHelper.h, que permite acceder a la información de la carga útil e incluye otras funciones relacionadas con la RTDB. Además, se define la red Wi-Fi disponible junto con su contraseña, se inserta la clave API del proyecto en Firebase y se proporciona la URL de este proyecto.

```
#include <SoftwareSerial.h>
#include "stdlib.h"
#include <Firebase_ESP_Client.h>
#include <Arduino.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include "addons/TokenHelper.h"
#include "addons/RTDBHelper.h"

#define WIFI_SSID "RESERVA SANGUARE"    //"Cristian"
#define WIFI_PASSWORD "sanguare01" //"crisqui..."
#define API_KEY "AIzaSyDM2jzBC5G-ViPBpYzHCNnj3UlYsFwRC9c"
#define DATABASE_URL "https://simevam-ii-default-rtdb.firebaseio.com/"
```

Se crean los objetos de datos, autenticación y configuración de Firebase. Además, se declaran las variables que se utilizan para el almacenamiento, así como una bandera que indica la disponibilidad para publicar los datos en la RTDB.

```
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

//Declaracion de variables
char inChar;
String string="";
bool signupOK = false;
```

Se inicia la conexión del NodeMCU a Internet. Si la conexión es exitosa, se muestra un mensaje en la consola de Arduino que indica "Conectado a IP: 192.168.XX.XX". En caso de fallo, se muestra un punto "." y se espera 3 segundos para intentar nuevamente la conexión. Es imprescindible asignar la clave API y la URL de la RTDB donde se desean publicar los datos.

```
void setup() {
  Serial.begin(9600);
  mySerial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED){
    Serial.print(".");
    delay(300);
  }
  Serial.println();
  Serial.print("Connected with IP: ");
  Serial.println(WiFi.localIP());
  Serial.println();
  config.api_key = API_KEY;
  config.database_url = DATABASE_URL;
```

Se verifican las credenciales ingresadas y, si son correctas, se accede a la base de datos. En caso contrario, se muestra un mensaje de error. Se declaran las variables donde se almacenarán los datos a publicar.

```
  if (Firebase.signUp(&config, &auth, "", "")){
    Serial.println("ok");
    signupOK = true;
  }
  else{
    Serial.printf("%s\n", config.signer.signupError.message.c_str());
  }
  config.token_status_callback = tokenStatusCallback;
  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);
}

String DatoString = "";
String SensorPath = "";
```

Dado que la función de bucle se ejecuta de forma continua, si la variable "inChar" es igual a "/", y si la variable "string" es igual a "RTD", "ORP", "EC", "OD" o "PH" (que son las abreviaturas definidas anteriormente para cada uno de los sensores), entonces se publica en la base de datos en tiempo real en la ruta indicada por la variable "sensorPath".

```
void loop() {
  if (mySerial.available()){
    inChar = mySerial.read();
    if(inChar!='\n' && inChar!='/'){
       string+=inChar;
    }
    if(inChar=='/'){
      if(string=="RTD"){
        SensorPath="App_SIREMEVAM/Sux6U6NwAAWIy451JcwK6d0Bxhj2/RTD";
      }else if(string=="ORP"){
        SensorPath="App_SIREMEVAM/Sux6U6NwAAWIy451JcwK6d0Bxhj2/ORP";
      }else if(string=="EC"){
        SensorPath="App_SIREMEVAM/Sux6U6NwAAWIy451JcwK6d0Bxhj2/EC";
      }else if(string=="OD"){
        SensorPath="App_SIREMEVAM/Sux6U6NwAAWIy451JcwK6d0Bxhj2/OD";
      }else if(string=="PH"){
        SensorPath="App_SIREMEVAM/Sux6U6NwAAWIy451JcwK6d0Bxhj2/PH";
      }
```

Si la variable "string" no coincide con ninguna de las etiquetas previamente mencionadas, entonces no se realiza la publicación en Firebase y se muestra el mensaje "FAILED" en la consola de Arduino.

```
      if (Firebase.ready() && signupOK){
            if(Firebase.RTDB.setString(&fbdo, SensorPath, string)){
              Serial.println("PASSED");
            }
            else {
              Serial.println("FAILED");
              Serial.println("REASON: " + fbdo.errorReason());
            }
        }

      string="";
    }
```

Hasta esta parte, los datos ya han sido publicados en Firebase, como se puede apreciar en la siguiente imagen. Cada vez que los datos cambian, estos cambios se reflejan en la RTDB. Este proyecto está titulado como VAMCI y está vinculado a una aplicación móvil.

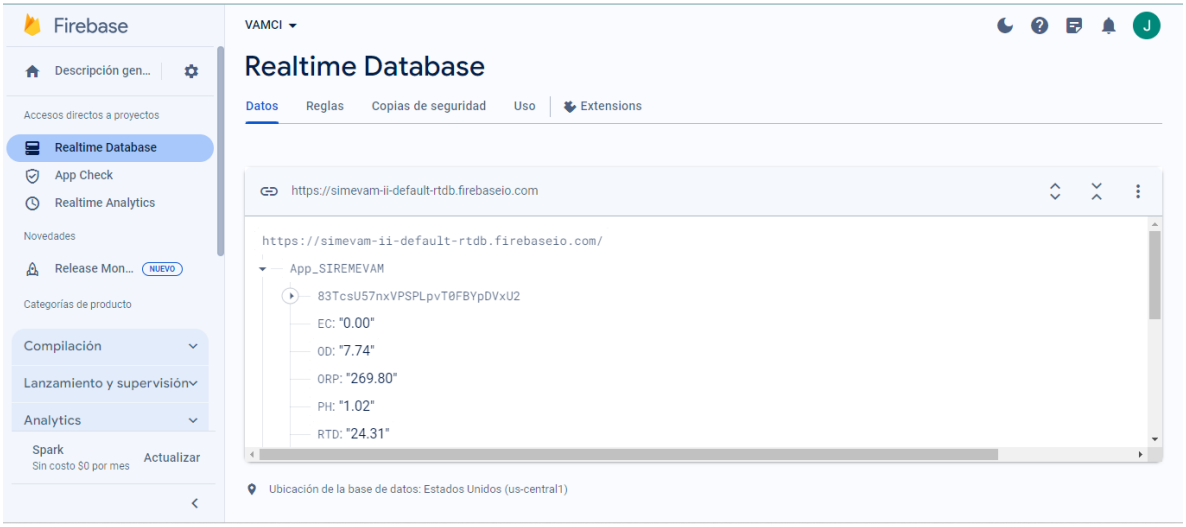


Figura 1. RTDB de Firebase

Para crear un proyecto en Firebase desde cero, visitar los siguientes links:

https://firebase.google.com/docs/firestore/quickstart?hl=es-419

https://www.youtube.com/watch?v=VKcav-7f2Qc

5.1.6. Almacenamiento en el repositorio .csv usando Google Sheets.

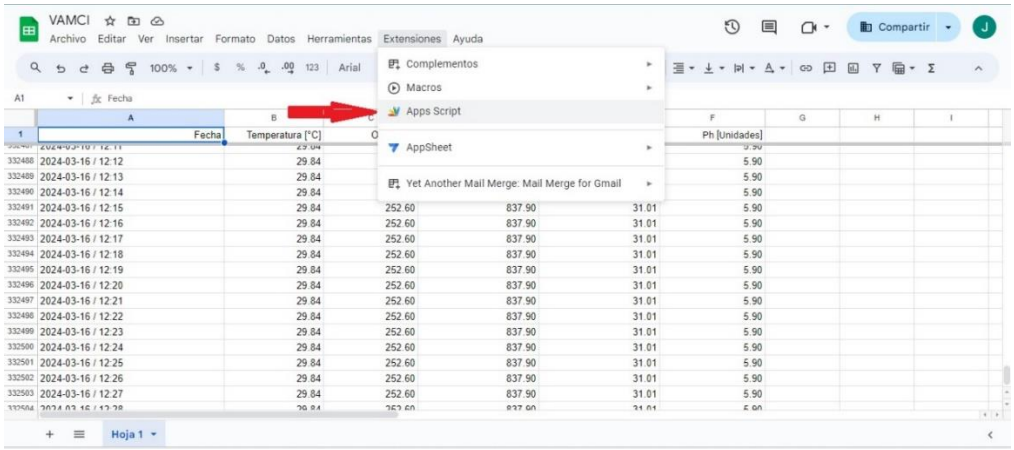


Figura 2. Hoja de cálculo VAMCI

Para almacenar los datos de los sensores, se crea una hoja de cálculo llamada VAMCI. Al utilizar Google Drive como servicio, se accede a las extensiones disponibles y se selecciona Apps Script. Al hacer clic en esta opción, se abre un editor de código donde se programa la manera de obtener los datos desde Firebase y el orden en que se mostrarán en el archivo .csv.

Es necesario asignar la URL desde la cual provienen los datos y la clave de autenticación de la base de datos. Luego, los datos se

ordenan en un arreglo con posiciones del 0 al 4, junto con la hora correspondiente que se extrae de la RTDB, llamando a las funciones previamente creadas.

Se crean funciones para calcular el año, el mes, el día, la hora y los minutos, ya que el aplicativo de visualización GEOVAM lo requiere de esta forma. Por último, se establece una función para extraer los datos necesarios de la RTDB, con condiciones para manejar las situaciones en las que la extracción de datos es exitosa y cuando hay fallos. Es importante recordar que Firebase genera un archivo JSON del cual se pueden extraer los objetos necesarios con las etiquetas de los sensores. A continuación, se muestra el código para una mejor comprensión.

```
url="https://vamci-default-rtdb.firebaseio.com/App_VAMCI/Sux6U6NwAAWIy451JcwK6d0Bxhj2.json?&
auh=tfCCZCoHUdi5KbZasl1tmww8k3rkcPG62Jb4DsT8"     //CHANGES like dht
function addProduct() {
  var sheet = SpreadsheetApp.getActiveSheet();
  var Time=Hora();
  var DatosVamci=[];

  for (var i = 0; i < 5; i++) {

      switch (i) {
        case 0:
          xpath="RTD";
          break;
        case 1:
          xpath="ORP";
          break
        case 2:
          xpath="EC";
          break;
        case 3:
          xpath="OD";
          break;
        case 4:
          xpath="PH";
          break;

      }
      DatosVamci[i]=IMPORTJSON();

  }
  sheet.appendRow([Time,DatosVamci[0],DatosVamci[1],DatosVamci[2],DatosVamci[3],DatosVamci[4]]);
}

function Hora() {

  var tiempo = new Date();

  var Month=tiempo.getMonth()+1;

  var MonthFin="";

  if(Month<10){
    MonthFin="0"+Month;
  }else{
    MonthFin=Month;
  }
```

```
  var Day=tiempo.getDate();
  var DayFin="";

  if(Day<10){
    DayFin="0"+Day;
  }else{
    DayFin=Day;
  }

  var objToday = new Date(),

  curHour = objToday.getHours() > 12 ? objToday.getHours()  : (objToday.getHours() < 10 ? "0" + objToday.getHours() : objToday.
  getHours()),
  curMinute = objToday.getMinutes() < 10 ? "0" + objToday.getMinutes() : objToday.getMinutes(),
  curSeconds = objToday.getSeconds() < 10 ? "0" + objToday.getSeconds() : objToday.getSeconds();

  var Time=tiempo.getFullYear() + "-" + MonthFin + "-" + DayFin + " " + "/" + " "  + curHour+ ":" + curMinute
  //Con Hora Local:
  return Time;
}
```

```
function IMPORTJSON(){
  try{
    // /rates/EUR
    var res = UrlFetchApp.fetch(url);
    var content = res.getContentText();
    var json = JSON.parse(content);

    var patharray = xpath.split("/");
    //Logger.log(patharray);

    for(var i=0;i<patharray.length;i++){
      json = json[patharray[i]];
    }
    //Logger.log(typeof(json));
    if(typeof(json) === "undefined"){
      return "Node Not Available";
    } else if(typeof(json) === "object"){
      var tempArr = [];

      for(var obj in json){
        tempArr.push([obj,json[obj]]);
      }
      return tempArr;
    } else if(typeof(json) !== "object") {
      return json;
    }
  }
  catch(err){
      return "Error getting data";
  }

}
```

Además, es posible controlar la frecuencia con la que se registran los datos en el archivo utilizando los activadores de ejecución del código disponibles en Apps Script. Estos activadores permiten seleccionar intervalos de tiempo, ya sea por minutos, horas, días, semanas, meses o años. Esto ayuda a ahorrar almacenamiento en la cuenta elegida para la creación del repositorio. A continuación, se muestran las imágenes que ilustran este procedimiento.

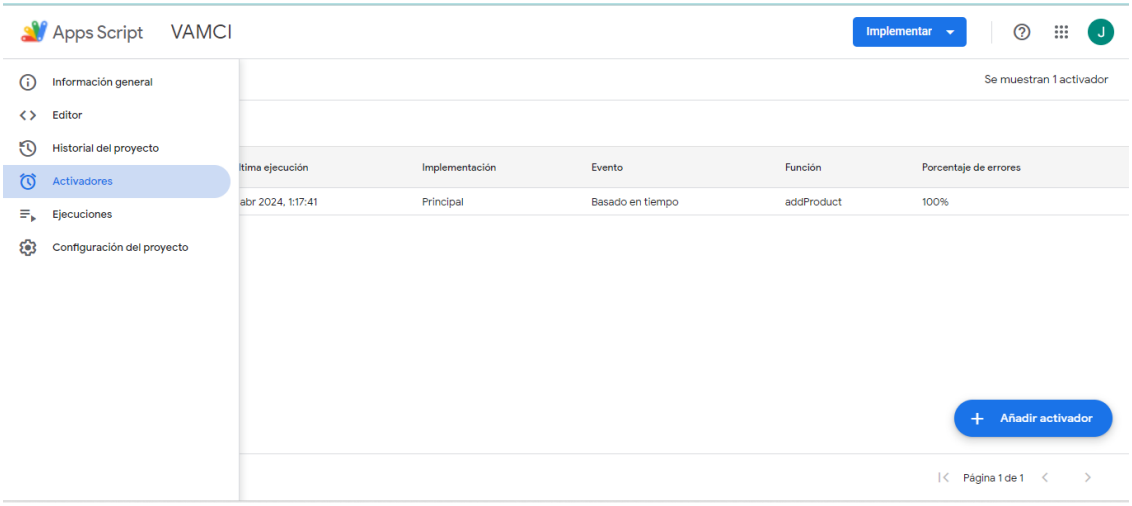
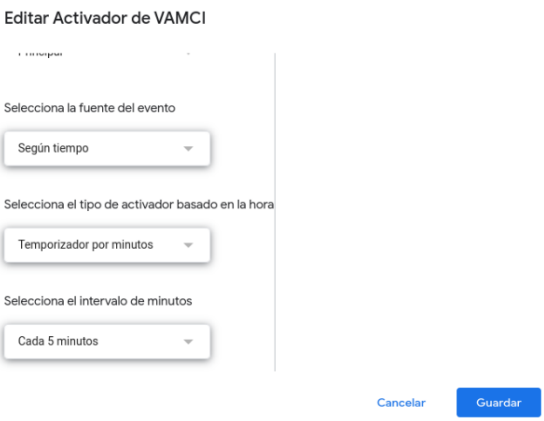


Figura 3. Menú principal de AppScript.



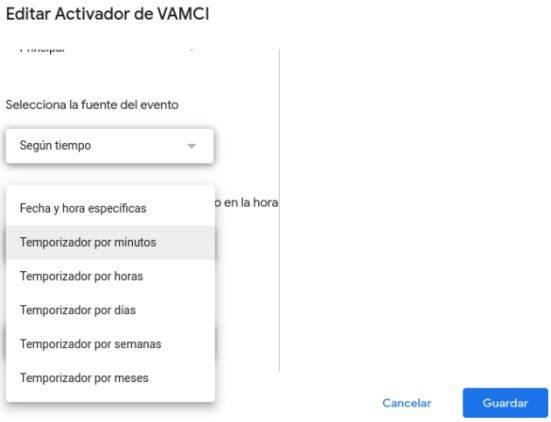
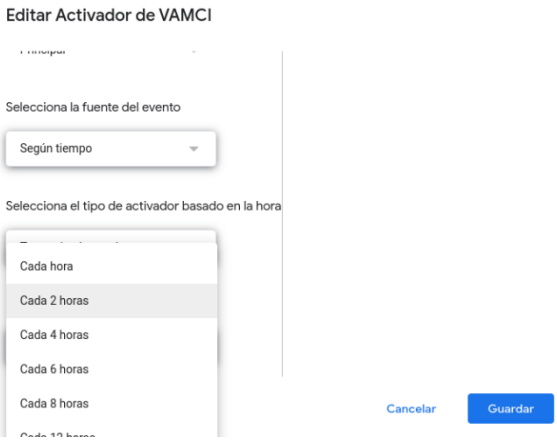


Figura 4. Opción de activador según tiempo. **Figura 5.** Opción de intervalos de Tiempo



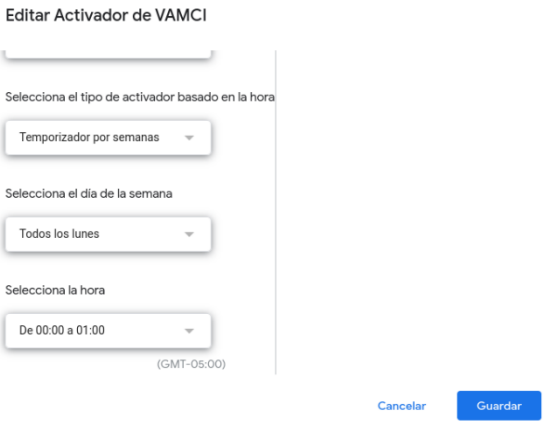
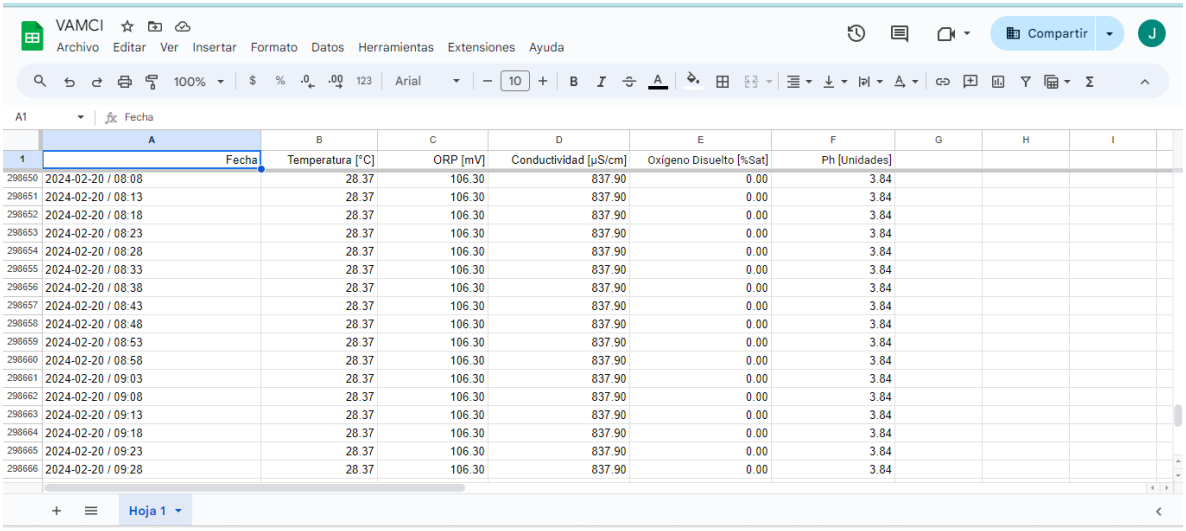


Figura 6. Activadores temporizados por hora. **Figura 7.** Activadores temporizados por semanas.

Estos son algunos ejemplos de los activadores disponibles en esta herramienta. Dependiendo del comportamiento del entorno donde se instale VAMCI, los responsables podrán decidir con qué frecuencia se registrarán los datos. Actualmente, los datos de los cinco sensores se registran con fecha y hora exactas cada 5 minutos. En la siguiente imagen se muestra cómo se encuentra el repositorio.



	Fecha	Temperatura [°C]	ORP [mV]	Conductividad [µS/cm]	Oxígeno Disuelto [%Sat]	Ph [Unidades]
298650	2024-02-20 / 08:08	28.37	106.30	837.90	0.00	3.84
298651	2024-02-20 / 08:13	28.37	106.30	837.90	0.00	3.84
298652	2024-02-20 / 08:18	28.37	106.30	837.90	0.00	3.84
298653	2024-02-20 / 08:23	28.37	106.30	837.90	0.00	3.84
298654	2024-02-20 / 08:28	28.37	106.30	837.90	0.00	3.84
298655	2024-02-20 / 08:33	28.37	106.30	837.90	0.00	3.84
298656	2024-02-20 / 08:38	28.37	106.30	837.90	0.00	3.84
298657	2024-02-20 / 08:43	28.37	106.30	837.90	0.00	3.84
298658	2024-02-20 / 08:48	28.37	106.30	837.90	0.00	3.84
298659	2024-02-20 / 08:53	28.37	106.30	837.90	0.00	3.84
298660	2024-02-20 / 08:58	28.37	106.30	837.90	0.00	3.84
298661	2024-02-20 / 09:03	28.37	106.30	837.90	0.00	3.84
298662	2024-02-20 / 09:08	28.37	106.30	837.90	0.00	3.84
298663	2024-02-20 / 09:13	28.37	106.30	837.90	0.00	3.84
298664	2024-02-20 / 09:18	28.37	106.30	837.90	0.00	3.84
298665	2024-02-20 / 09:23	28.37	106.30	837.90	0.00	3.84
298666	2024-02-20 / 09:28	28.37	106.30	837.90	0.00	3.84

Figura 8. Datos registrados cada 5 minutos.

5.2. Código app en Kodular de la app móvil VAMCI.

La aplicación VAMCI se desarrolló utilizando Kodular, un sitio web que facilita la conversión de ideas en aplicaciones móviles de manera sencilla. Debido a que Kodular utiliza una metodología de programación en bloques, resulta más entretenido plasmar las ideas de manera visual.
Para crear un proyecto en Kodular desde cero, se puede seguir el tutorial disponible en el siguiente enlace:
https://www.youtube.com/playlist?list=PLqH5eJjxXS8SYsyyJ56CCMbmsxGVwvWVh

5.2.1. Programación screen inicial y Login.

La aplicación empieza con una pantalla que muestra el logo de la aplicación, el cual incluye el nombre del proyecto y el nombre de los grupos de investigación. Este diseño se implementa de la siguiente manera:

Figura 9. Diseño de la pantalla “Inicial”.

Como se puede ver en los bloques, se arrastra un bloque de tipo animación que carga el logo y lo hace desplazarse horizontalmente durante 3 segundos. También se incluye un banner verde que simula el paso de los segundos para abrir el siguiente screen.
El siguiente screen es el de inicio de sesión (Login), donde se anima el logo para que se desplace hacia arriba y hacia abajo en un movimiento de resorte hasta que regrese a su posición original. Además, se incluyen dos botones: uno para registrarse en la aplicación del sistema y otro para iniciar sesión si el usuario ya está registrado.
Como se puede ver en los bloques, se utiliza un bloque para animar el logo, un procedimiento para cerrar la pantalla y bloques de control para abrir las pantallas de Registro e Iniciar sesión, dependiendo de la opción seleccionada por el usuario.

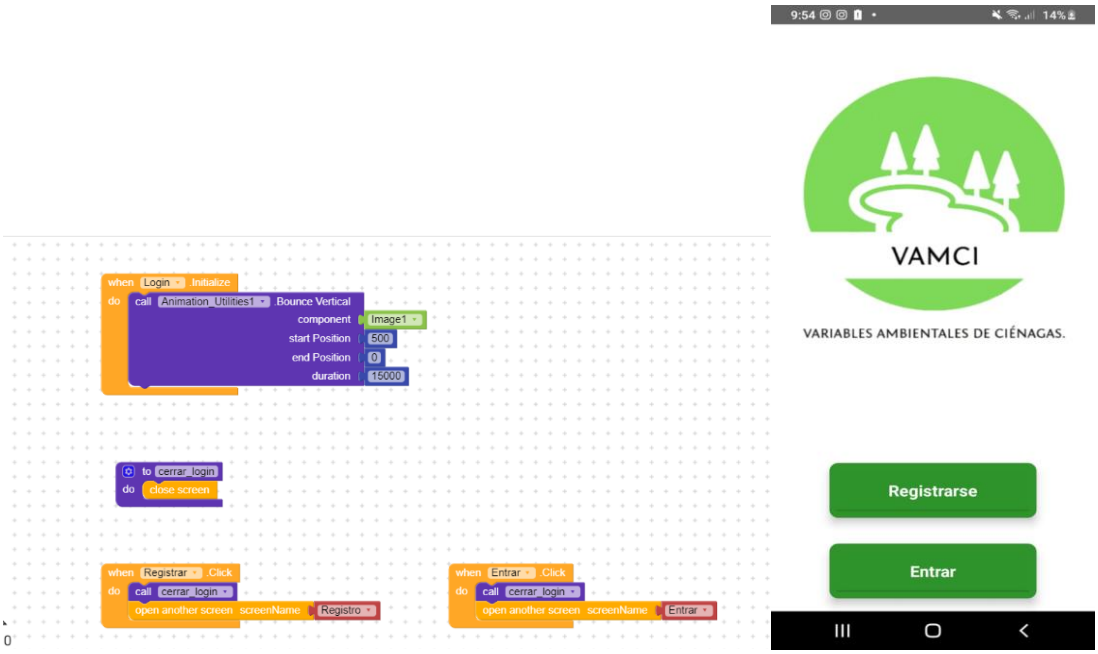


Figura 10. Diseño de Pantalla de “Login”.

5.2.2. Registro de usuarios nuevos.

El siguiente screen es el de Registro de usuario nuevo. Aquí se invocan los métodos para que el teclado sea visible al presionar alguno de los campos por completar. Se programan las validaciones, por ejemplo, que en el campo "celular" solo se admitan caracteres numéricos o que en el campo "correo", el texto ingresado debe contener el carácter "@" para que sea válido. Además, se programa el bloque para que toda la información ingresada se almacene en la base de datos en tiempo real de Firebase. También se programa una barra de progreso para que sea visible al momento de cargar la foto, si así se desea. A continuación, se muestra lo mencionado:

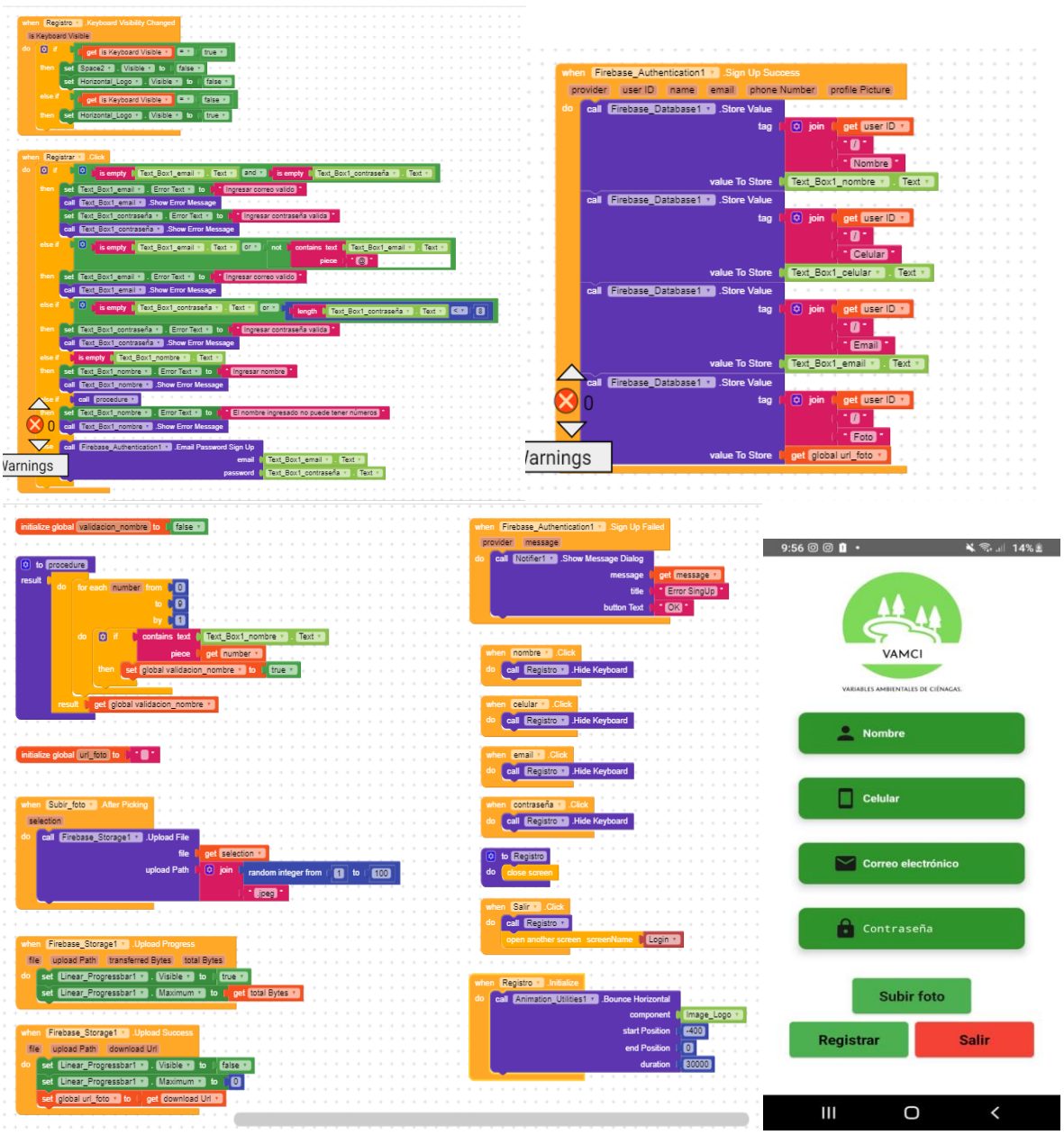


Figura 11. Diseño de pantalla de “Registro”.

5.2.3. Programación screen Entrar.

El siguiente screen que se ha programado es el de "Entrar", donde se han realizado las validaciones del correo ingresado y la contraseña previamente establecida en el screen anterior. Además, se han implementado los métodos para hacer visible el teclado cuando se presiona alguno de los campos.

Se ha incorporado el bloque de autenticación de Firebase. Cuando las credenciales son correctas, se muestra un mensaje que indica "Autenticación correcta". En caso contrario, se despliega un mensaje indicando "Ingresar correo válido" o "Ingresar contraseña válida".

Además, se ha configurado la acción de pasar al screen "Perfil" cuando se presiona el botón "Entrar", y de regresar al screen de "Login" cuando se selecciona el botón "Salir". Se ha programado también el logo para que gire mientras se ingresan las credenciales, con una duración de 25 segundos.

A continuación, se muestran los bloques y la interfaz correspondientes.

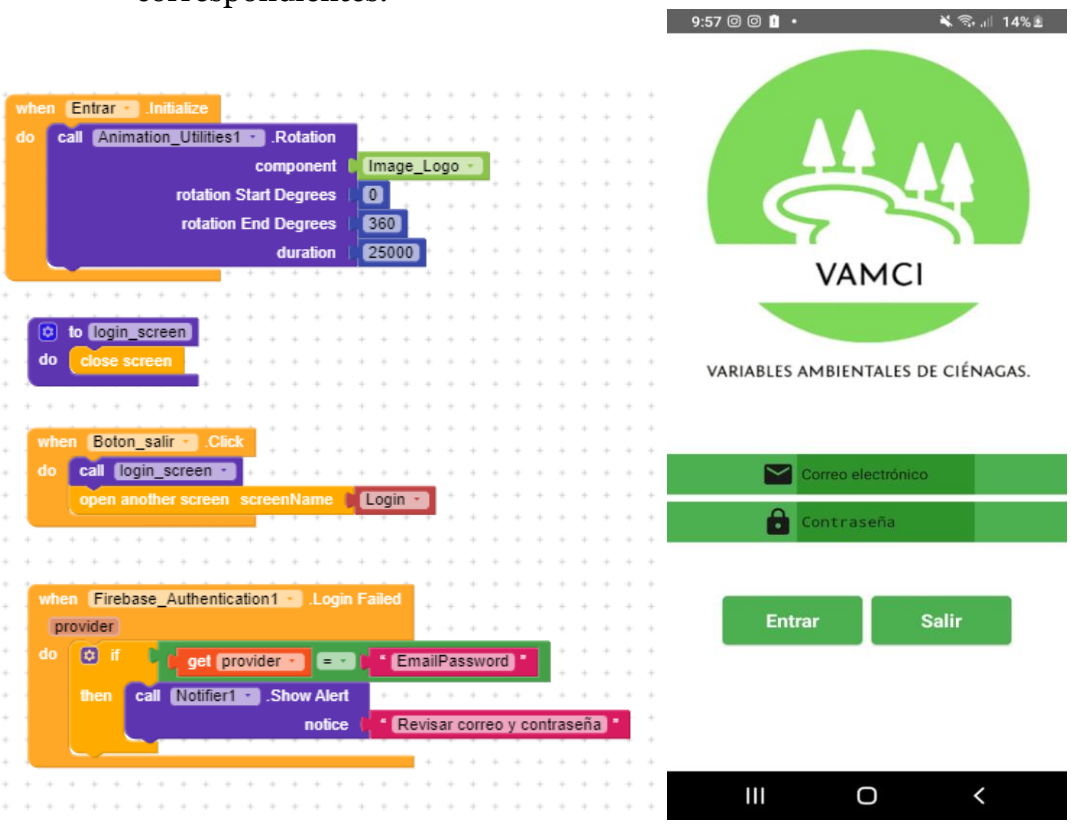
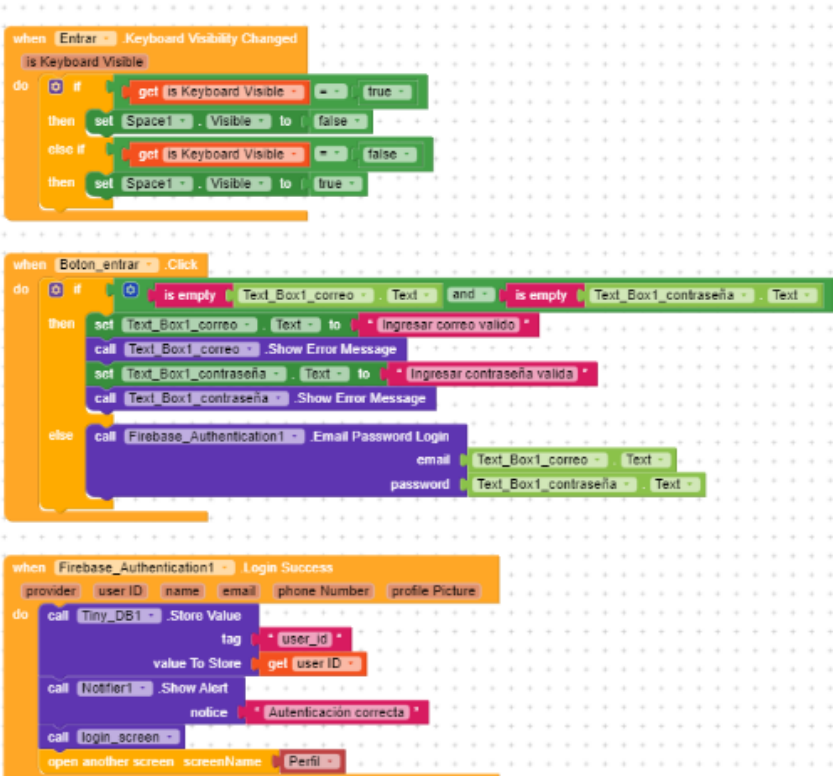
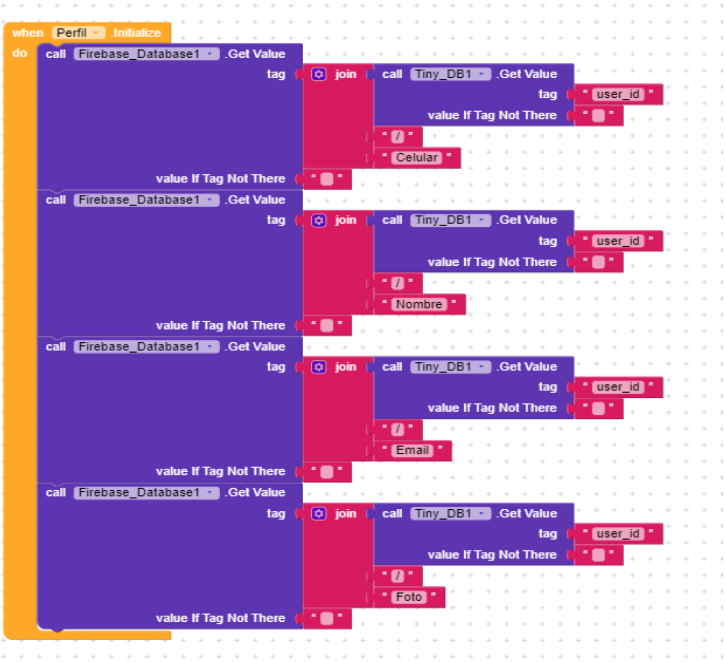


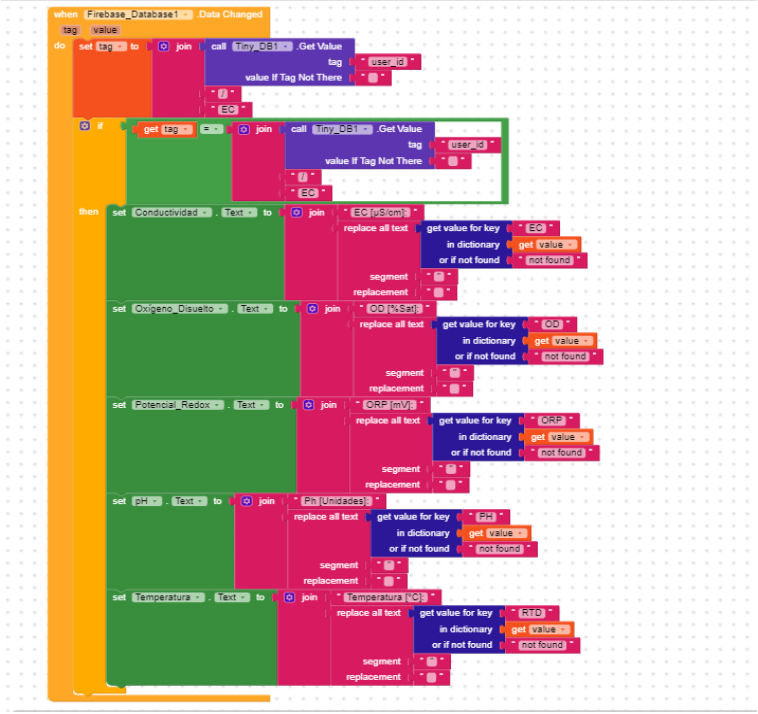
Figura 12. Diseño de pantalla “Entrar”.



5.2.4. Interfaz del perfil y consulta de datos.

Cuando se accede al screen "Perfil", se recuperan los datos ingresados anteriormente: la foto, el nombre, el correo electrónico y el número de teléfono que se encuentran almacenados en Firebase y que fueron proporcionados durante el registro en la aplicación. Además, se obtienen los valores de los sensores y se muestran en los correspondientes "card_view", junto con sus unidades para facilitar la comprensión.
Si se presiona alguno de los "card_view" donde se muestran los datos en tiempo real, estos son consultados directamente en la RTDB y se abre el screen "Históricos". Por otro lado, si se selecciona el botón "Salir", se vuelve al screen de "Login". Todo este proceso se muestra a continuación.


when Perfil .Initialize
do
call Firebase_Database1 .Get Value
tag
join
call Tiny_DB1 .Get Value
tag
user_id
value If Tag Not There
/
Celular
value If Tag Not There
call Firebase_Database1 .Get Value
tag
join
call Tiny_DB1 .Get Value
tag
user_id
value If Tag Not There
/
Nombre
value If Tag Not There
call Firebase_Database1 .Get Value
tag
join
call Tiny_DB1 .Get Value
tag
user_id
value If Tag Not There
/
Email
value If Tag Not There
call Firebase_Database1 .Get Value
tag
join
call Tiny_DB1 .Get Value
tag
user_id
value If Tag Not There
/
Foto
value If Tag Not There


when Firebase_Database1 .Data Changed
tag
value
do
set tag to
join
call Tiny_DB1 .Get Value
tag
user_id
value If Tag Not There
/
EC
if
get tag =
join
call Tiny_DB1 .Get Value
tag
user_id
value If Tag Not There
/
EC
then
set Conductividad . Text to
join
EC [uS/cm]
replace all text
get value for key
EC
in dictionary
get value
or if not found
not found
segment
replacement
set Oxigeno_Disuelto . Text to
join
OD [%Sat]
replace all text
get value for key
OD
in dictionary
get value
or if not found
not found
segment
replacement
set Potencial_Redox . Text to
join
ORP [mV]
replace all text
get value for key
ORP
in dictionary
get value
or if not found
not found
segment
replacement
set PH . Text to
join
Ph [Unidades]
replace all text
get value for key
PH
in dictionary
get value
or if not found
not found
segment
replacement
set Temperatura . Text to
join
Temperatura [C]
replace all text
get value for key
RTD
in dictionary
get value
or if not found
not found
segment
replacement

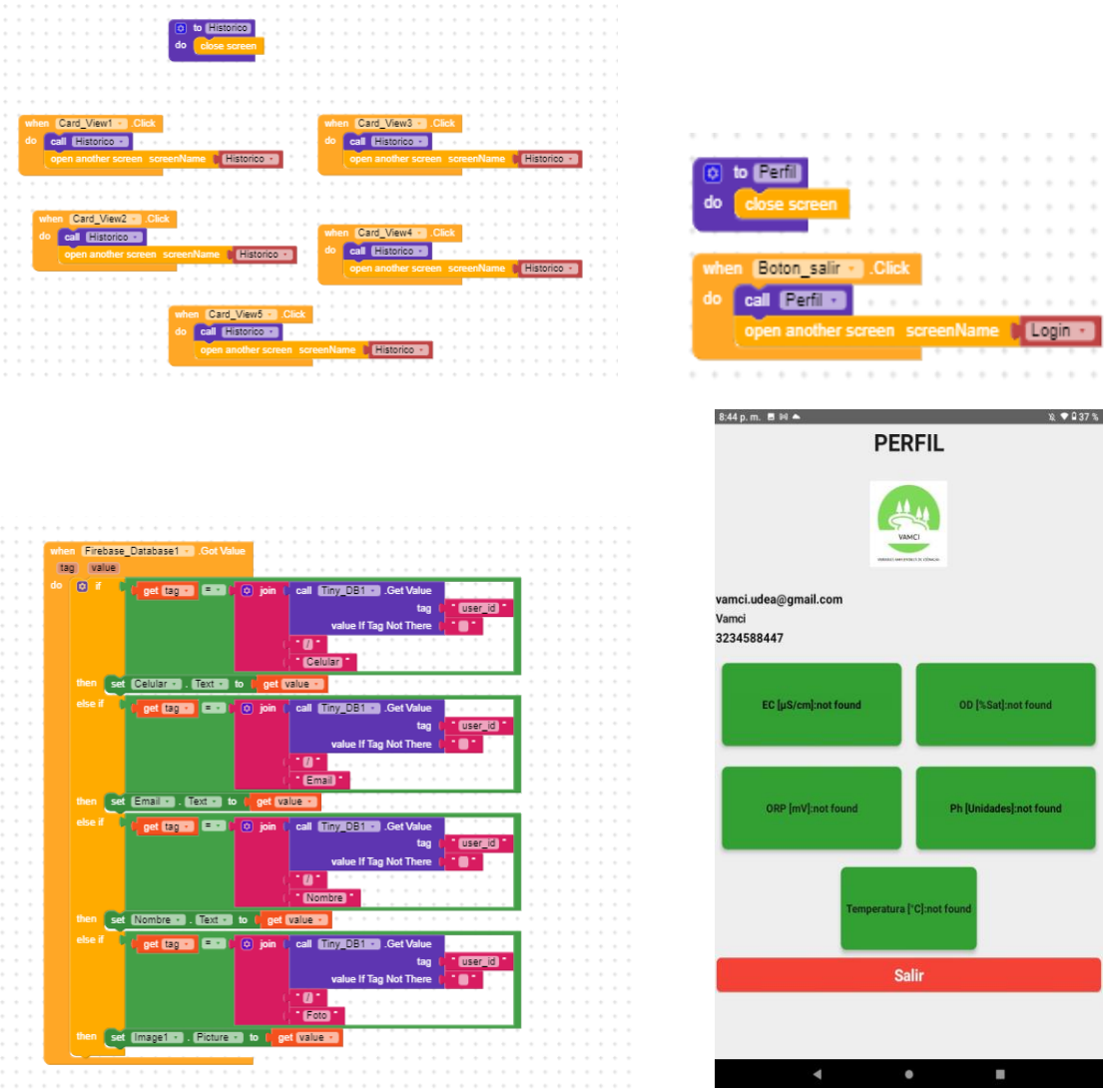


Figura 13. Interfaz del perfil de usuario.

5.2.5. Consulta de históricos en GEOVAM.

Como se mencionó anteriormente, al presionar cualquiera de los botones verdes que muestran las mediciones en tiempo real de los sensores, se abrirá el screen "Históricos". En este screen, se muestra un título y dos botones. El botón verde, denominado "GEOVAM", al ser presionado, abre el navegador instalado en nuestro smartphone y accede al hipervínculo del sitio web geovam.udea.edu.co, donde se pueden visualizar directamente los gráficos históricos de cada una de las variables. Por otro lado, si se presiona el botón "Salir", se vuelve al screen "Perfil". Si se desea salir de la aplicación, simplemente hay que presionar el botón de retroceso en nuestro teléfono inteligente.

A continuación, se muestran los bloques y la interfaz de este screen, así como el acceso desde el móvil a GEOVAM.

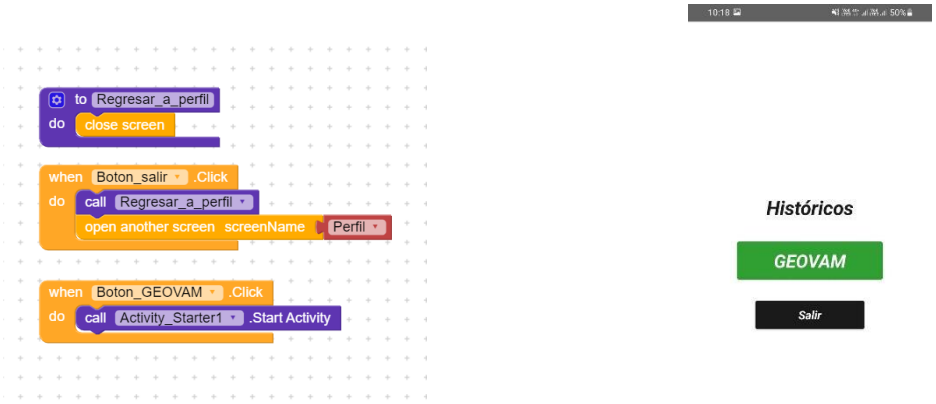


Figura 14. Pantalla de acceso a GEOVAM.

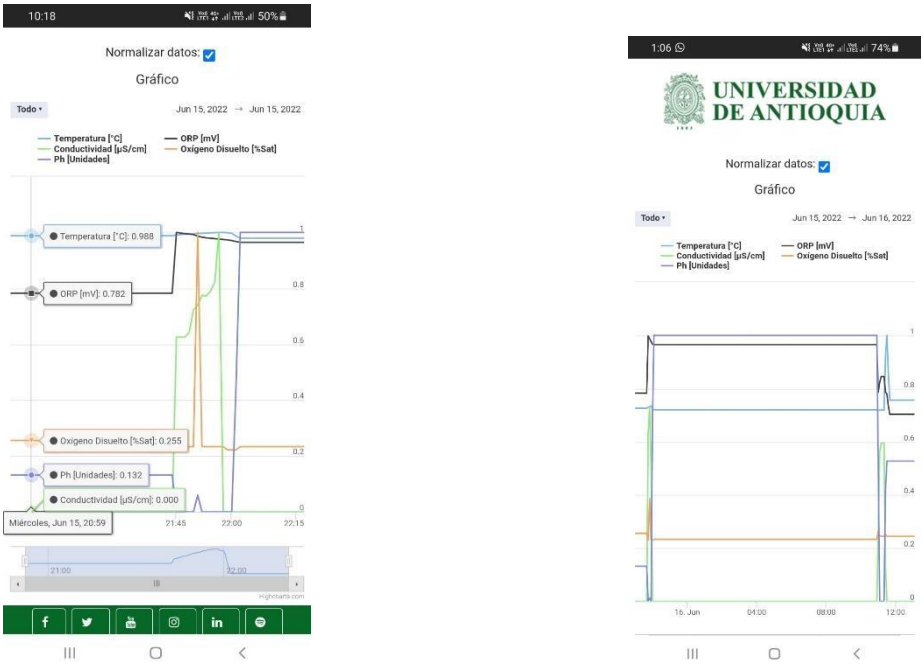


Figura 15. Gráficos históricos de las variables.

6. REFERENCIAS.

[1] "IEEE Standard for Software User Documentation," *IEEE Std 1063-2001*, pp. 1–24, 2001

[2] Arduino. (2022). *Arduino Mega 2560*. https://arduino.cl/arduino-mega-2560/

[3] Atlas Scientific | Environmental Robotics. (2022). *ENV-SDS Kit*. https://atlasscientific.com/kits/env-sds-kit/

[4] Atlas Scientific | Environmental Robotics. (2022). *Whitebox T2*. https://atlasscientific.com/electrical-isolation/whitebox-t2/

[5] Heltec. (2022). *Heltec Lora 32 versión 2*. https://heltec.org/product/wifi-lora-32-v2/

[6] Mkelectrónica. (2022). Placa *NodeMCU 1.0*. https://mkelectronica.com/producto/modulocontrolador-nodemcu-1-0/

[7] Ríos, J. (2021). *GEOVAM*. Universidad de Antioquia. https://geovam.udea.edu.co/

[8] Quintana Valencia, C. (2022). *Sistema remoto para la medición y registro de variables ambientales* [Trabajo de grado profesional]. Universidad de Antioquia, Medellín, Colombia.

[9] Brayn Wills. (2024). "Technical Manual: What, Types & How to Create One? (Steps Included) - Knowledgebase Blog." https://www.proprofskb.com/blog/write-technical-manual/